Condition Variables (CV)

- Locks help in protecting the critical section by providing mutual exclusion property. During unlock it wakes up one or more threads waiting for the lock.
- However, a thread might be interested in entering the critical section only when a certain event happens or a condition is satisfied. This notion of wakeup on condition is provided by condition variables.
- Condition variables help in identifying state change of the critical section. E.g. A thread waiting for memory to be allocated should be woken up only a free event happens.

CV primitives

- Initialize
  - Conditional variable is a queue of waiting threads that is also associated with a mutex variable to protect the critical section
    
    ```
    pthread_cond_t c = PTHREAD_COND_INITIALIZER;
    pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
    ```
- wait(): waiting for any change in state for the shared region
  - The mutex lock should be held when issuing a wait on the conditional variable
  - Note that mutex is also passed along with the wait statement. The caller is put to sleep and the mutex is released by the wait call.
  - The lock will be held on resume i.e. return from the wait call.
  - wait returns only when a signal happens and the waiter is able to grab the mutex during wait
    ```
    pthread_mutex_lock(&m);
    // wait for any state change in a loop
    pthread_cond_wait(&c, &m);
    pthread_mutex_unlock(&m);
    ```
- signal(): notify to the waiters that state of shared region has changed
  - The mutex lock is held when changing the critical region
  - Notify waiters through signal statement when the state is to be modified
  - Signal guarantees that it wakes up at least one waiter thread if there are any (it just provide a hint that state has changed and does not convey any other information)
  - Upon unlock, the newly woken waiter thread could grab the mutex lock during signal
    ```
    Pthread_mutex_lock(&m);
    // change state
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
    ```
Signal broadcast

- The native signal functions wakes up only one waiter thread
- But different waiters could be expecting different states. E.g. Waiter 1 expects shared region to be in state A, waiter 2 expects shared region to be in state B.
- It is not clear which waiter should receive the signal
- New signal interface is supported to wake up all the waiter threads. Now, it is the responsibility of the waiter threads to ensure that the critical region is in the expected state before proceeding. Else go back to sleep by waiting on the conditional variable.

Examples

- Notification when a child exits
- Producer/Consumer or Bounded-Buffer problem
- Memory allocation